Public Transport (2022) 14:27–61
https://doi.org/10.1007/s12469-022-00295-3

ORIGINAL PAPER

Resolution of coupling order and station level constraints in train unit scheduling

Li Lei[1] · Raymond S K Kwan[1] · Zhiyuan Lin[2] · Pedro J Copado-Mendez[3]

Accepted: 17 February 2022 / Published online: 11 May 2022

Abstract

Train unit scheduling assigns vehicles to cover all trips of a fixed timetable satisfying constraints such as seat demands. With a two-phase approach, this problem is first solved in Phase I as an integer multi-commodity flow problem. Train stations are simplified as single points and coupling orders of train units are left undetermined. In this paper, platforms and their layouts at the stations are restored to complete a fully operable schedule, defined as Phase II. An adaptive approach expanding Phase I to Phase II is proposed. The logistics of (de-)coupling operations, coupling orders and re-platforming are determined in detail to prevent unit blockage where possible, particularly focusing on developing a schedule with conflict-free coupling orders. If unresolvable station level conflicts still exist, the process loops back to Phase I with addressed Phase II constraints to avoid identified conflicts. Thus, the schedule is iteratively improved until it is fully operable.

Keywords Train unit scheduling · Coupling order · Station constraints · Resolution

✉ Li Lei
scll@leeds.ac.uk; 1071577471@qq.com

Raymond S K Kwan
r.s.kwan@leeds.ac.uk

Zhiyuan Lin
Z.Lin@leeds.ac.uk

Pedro J Copado-Mendez
pcopadom@uoc.edu

[1] School of Computing, University of Leeds, Leeds, UK

[2] Institute for Transport Studies, University of Leeds, Leeds, UK

[3] IN3-Computer Science Department, Universitat Oberta de Catalunya, Castelldefels, Spain

1 Introduction

A *train unit* (TU) is a kind of passenger rolling stock that has fixed carriages and integrated engines that can move in either direction. Train units are commonly used in railway passenger transport as opposed to traditional locomotive trains. They are classified into different *types* by characteristics such as power source, number of carriages, and number of seats. Currently 17 franchises managing a certain number of train units are operating the UK railway (Wikipedia 2019). A timetable describes a set of *trips* with fixed routes (origin, destination and intermediate stations) and timings (clock times of arrival and departure). Train units are assigned to serve trips with sufficient capacities to meet passenger demands, i.e., a trip is served by a single unit or multiple units defined as a *unit block*. To satisfy different seat demands of trips, *coupling operations* may be involved to form a longer unit block, or *decoupling operations* to split a longer formation into shorter unit blocks. Another purpose of coupling and decoupling operations is to redistribute unit resources across the network. This train unit scheduling problem (TUSP) is shown to be NP-hard (Cacchiani et al. 2010; Lin and Kwan 2017). If a trip is served by multiple units of different types, the sequence of the units within the serving unit block is important, in particular when the unit block is involved in coupling or decoupling operations. In the UK, the coupling and decoupling operations are usually executed at station platforms. As the movement of unit blocks is strictly restricted by tracks, station layouts and how they connect with other stations are key factors to consider for removing coupling-order conflicts.

The TUSP can be solved in two phases: network level scheduling and station level scheduling (Lin and Kwan 2014). The network level scheduling problem is studied by Cacchiani et al. (2010) and Lin and Kwan (2013). In real-world railway operations, there are large numbers of timetabled trips to be scheduled with complex network constraints, for instance passenger demands, fleet size, unit type compatibility, turnaround time, making train unit scheduling at the network level a challenging problem. Solving the TUSP including station level constraints is even much harder. Thus, reasonable simplifications are applied. Stations are considered as single points where all train unit movements required for trip connections are assumed feasible no matter what schedule is formed at the network level. The station level constraints are important when a network level schedule is supposed to be implemented on physical rail infrastructure. Therefore, reconsidering station level constraints in Phase II is vital to avoid operational blockage during the station operating process and to determine conflict-free coupling orders, which makes the network scheduling results fully operable in practice.

The basic Phase I model optimizes train unit assignment at the network level by branch-and-price and tentatively allocates unit blocks to trips and their connections. Its solution is incomplete since coupling orders are not determined and the feasibility of tentative linkages among trips has not been verified. This incompleteness may cause severe operational conflicts at stations. Phase II is to take a further step of station level resolution and to fix the incompleteness. Station level constraints would be very complex and difficult to generalize in an all

encompassing TUSP model; therefore, they are ignored in Phase I. Phase I has the benefit that its solution may have already automatically satisfied many of the station level constraints even though they have not been explicitly modeled. In other words, the station level conflicts in a Phase I solution could often be sparse. In this paper, an adaptive approach of incrementally inserting constraints for Phase II eliminating station level conflicts identified in a Phase I solution is proposed. The main loop of this method is between a Phase I core solver (*RS-Opt*, developed by Lin and Kwan (2016a)) and Phase II including the process of station level conflict detection and coupling order assignment. This approach resolves the station level constraints and assigns feasible coupling orders for trips served by multiple units. A new solution will be sought if Phase II has encountered unresolvable conflicts, and newly formulated critical station constraints will be fed back to the core solver to eliminate those inoperable conflicts. Feasible coupling orders are assigned simultaneously with the conflict detection process.

The structure of this paper is organized as follows. Section 2 gives a literature review. Section 3 introduces the problem of train unit scheduling with station constraints. Section 4 describes a mathematical model and discusses the complexity of station level constraints. Section 5 formally defines the coupling order and its propagation boundaries. Section 6 illustrates the proposed adaptive approach and methods of conflict detection and coupling order assignment. The experimental results are shown in Sect. 7. Conclusions and ongoing research are discussed in Sect. 8.

2 Literature review

Huisman et al. (2005) survey passenger railway transportation operational research focusing on European cases, where train unit planning is classified as two levels. The central level covers the entire network operations such as timetabling, train unit scheduling, circulation, etc. The local level focuses on the local-size-impact operations including shunting, platform assignment, routing of train units at a certain station, etc. The train unit scheduling problem is also addressed as two levels (Lin and Kwan 2014) and (Kwan et al. 2017). The network level aims to solve train sequencing and fleet assignment where coupling and decoupling activities are important to satisfy passenger demands. The station level copes with finalizing the operational plan and train unit shunting within stations/depots.

For the network level problem, Cacchiani et al. (2010) consider the basic constraints of passenger demands, coupling upper bounds and additional constraints of maintenance and overnight balance. Two inter-convertible ILP (integer linear programming) formulations are established based on a directed acyclic graph as described in Cacchiani et al. (2013a). After solving the corresponding relaxed LP, a heuristic is designed to further solve the network level problem. The result shows that it is crucial in finding feasible solutions in many tested instances since the problem is regarded as very difficult. Thus, another heuristic method based on Lagrangian relaxation is developed by Cacchiani et al. (2013b) to improve the performance. A more comprehensive model with many real-world constraints is proposed by Lin and Kwan (2013) to describe the network flow problem in

the UK. A branch-and-price method called *RS-Opt* is developed (Lin and Kwan 2016a). A heuristic wrapper for *RS-Opt* called SLIM is proposed in Copado-Mendez et al. (2017) to solve larger problem instances. In practice, most train operators in the UK consider different levels of capacity provisions, such as peak and off-peak. To achieve this, Lin et al. (2017) study the train unit scheduling problem with bi-level capacity requirements, in which a new integer multi-commodity flow model guided by historic capacity provisions is proposed. The computational experiments on real-world data have shown this methodology to be effective.

A timetable defines arrival/departure times and platforms for each trip, while a train unit schedule assigns unit blocks to serve each trip. At the station level, given a timetable with a train unit schedule, a shunting schedule guarantees that the assigned unit blocks are operable at the fixed timings and platforms of stations. Tomii et al. (1999) consider the station shunting problems as a resource constrained scheduling problem and divide it into two subproblems: resource allocation and shunting time decision. A two-stage-search algorithm is proposed to solve this problem combining with probabilistic local search and Program Evaluation Review Technique (PERT). Freling et al. (2005) and Kroon et al. (2008) consider the train unit shunting problem at a station as two subproblems: matching problem and parking problem. Freling et al. (2005) describe a set-partitioning ILP model to solve these two subproblems separately, which prevents capacity overflow and unit blockage at all sidings and minimizes the number of coupling/decoupling operations. This model is solved by root-only column generation combined with a branch-and-bound strategy. Since the matching and parking problems are connected to each other, Kroon et al. (2008) propose an integrated approach with four models considering both dead-end and through types of sidings, thus, the global optimality is guaranteed. The idea of virtual tracks is introduced to reduce the problem size. Computational experiments on two stations of the NSR network have been done.

Train unit scheduling at the network level and station level are interrelated. Models for the network level and the station level are proposed (Lin and Kwan 2014), in which the two levels can communicate through arc variables. Conceptually, the re-matched linkages at the station level model are encouraged if they are also chosen by the network level. Ideally, a good quality and operable solution at both levels can be achieved through this communication. Lei et al. (2017) analyze the potential station shunting issues caused by a solution at the network level without the determination of coupling orders. In addition, a branch-and-cut algorithm of connecting flow level and depot shunting is proposed by (Haahr and Lusby 2017) with the simplified assumption that tracks are all dead-end. For the train unit scheduling problem, the network level assigns train unit types and quantities to each timetabled trip. Once an assignment has been found from the simplified model temporarily ignoring station level constraints, there are two operational aspects left to be further determined. The first aspect is the unit coupling order in multi-type trips, which has no impact on the network flow but must be finalized to prevent blockages at stations. The second aspect is the tentative linkage implication restricted by station layouts. This research focuses on these two points to make the train unit scheduling solution operable at both the network and station levels which has not been scrutinized in literature.

3 Problem description

The train unit scheduling problem with station constraints is to find a conflict-free schedule at both the network and station levels and to minimize the general operational costs. The network level concerns train unit assignment to satisfy all the trips fixed in a timetable with respect to a set of constraints such as passenger demands, turnaround time, and unit type compatibility. Given a network level solution, this section mainly describes the relevant aspects concerned at the station level, including coupling order, linkage slack time, crossing linkages, platform type and capacity, re-platforming shunting, interchangeability between train units of the same type, and other features.

3.1 Coupling order

A coupling order is defined as the permutation of the units within a coupled train unit block. Given a block with n units, the coupling orders of this block have $n!$ possibilities. Considering the station level infrastructure, not all of these possibilities are operable because many coupling/decoupling activities can only be operated in a specific order under the station shunting environment.

Train stations are not isolated in a rail network but connected by routes. Running trips concatenate the operations at the starting, intermediate, and terminating stations. A coupling order is initially formed at the origin station of a trip, and its influence is not confined within the origin but can be propagated to other stations by running trips and the connections among trips, defined as *coupling order propagation*. Thus, a coupling order formed at a specific station must be also operable at other influenced stations at any time.

If a conflict caused by a coupling order arises at a station, additional operations, e.g., re-ordering shunting, must be adopted to fix this invalidity. This not only increases operational costs but may also disturb/delay other trips. Thus, coupling order decisions at stations can be sophisticated.

Usually, no coupling or decoupling operation happens at intermediate stations, but one feature needs to be captured while moving directions are considered. If the arrival and departure directions of a coupled unit block are opposite at some platforms of intermediate stations, the front and rear of the unit block will be reversed; otherwise, the coupling order keeps the same. This is defined as *coupling order reversal en-route*. Thus, a trip may have different unit sequences at its origin, intermediate stations, and destination.

3.2 Linkage slack time

The slack time of a linkage (directed arc in an directed acyclic graph (DAG), see Sect. 4.1) is defined as the time gap between the departure of its head node trip and the arrival of its tail node trip, which is the total time available for the station level manoeuvre such as coupling/decoupling, reordering, re-platforming, etc. At the

network level, the time consumed by those auxiliary station level operations are not properly accounted for, i.e., insufficient slack time may invalidate the network level schedule. A linkage is locally feasible at a station if there is sufficient slack time to accomplish essential station shunting movements for the purpose of avoiding a station conflict.

$$\Delta t = t_a^{slack} - t_a^{trt} - t_a^{sta} \geq 0, \ \forall a \in A \tag{1}$$

Hence, the slack time of each arc in a network level schedule must be verified with Eq. (1), where t_a^{slack}, t_a^{trt}, t_a^{sta} represent slack time, standard turnaround time, and station operational time corresponding to arc a, respectively. Δt must be not less than 0; otherwise, the arc is infeasible.

3.3 Platform type and capacity

For daytime scheduling, coupling and decoupling operations are often executed at platforms in the UK. There are two main types of platforms: dead-end platforms and through platforms. A dead-end platform can only be approached from one end of the track, i.e., the train units accumulated on it must follow the first-in-last-out (FILO) rule. The moving directions of approaching and leaving a dead-end platform must be opposite to each other. On the other hand, a through platform can be reached via both ends, which has a more complex utilization in the real-world railway management. It is common in the UK railway industry to logically divide a through platform into a few sub-platforms, for instance, Cardiff Central Station through platforms 3 and 4 are used as 3A, 3B, and 4A, 4B, respectively. This logical dividing gives through platforms a lot more flexibility but also increases the problem complexity. In practice, the trips are operated by many train operating companies, and platforms are often divided by train operation companies at a station. In this research, we only consider the trips operated by a single company. At the network level, the coupled unit upper bound and carriage upper bound for each trip are considered based on the limitation of the physical length of the corresponding platform. At the station level, the unit accumulation on the platform with time is considered to assure that the units parking at the same platform will not invalidate the schedule.

3.4 Crossing linkages

To illustrate, suppose four single unit trips i, j, m, n are linked at a through platform at station B with no problem of unit type compatibility and seat capacities. Two possible network level solutions are constructed, as shown in Fig. 1, referred as the first-in-first-out (FIFO) connection and the FILO connection. However, these two solutions may turn out be infeasible while considering the unit-block moving direction on the restricted station tracks. Figure 2 shows a possible parking position of these two units at a platform before the departure of j and n. Concerning the departure time of j and n, solution (1) will be feasible if and only if j departs to the "up direction" and the solution (2) will be feasible if and only if j departs to the "down direction". Otherwise, a crossing occurs as the unit assigned to n blocks the

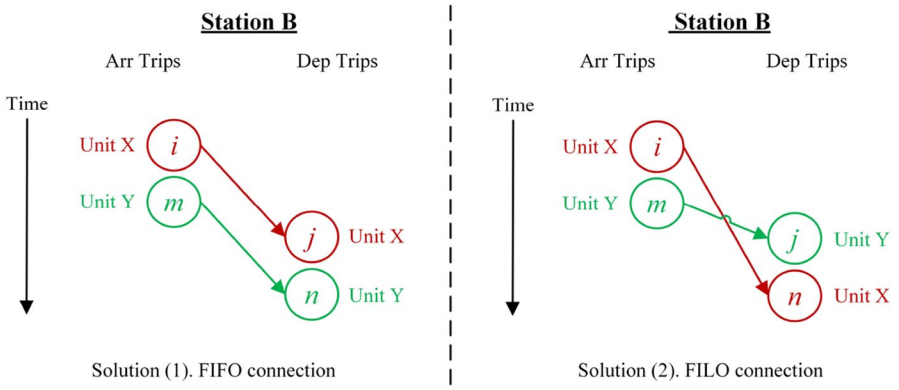

Fig. 1 Two possible solutions

Fig. 2 Example snapshot of two units prior to their next assigned departures

unit serving j, which is supposed to travel earlier than n. If no station information is introduced in the model, the feasibility of tentatively assigned arcs cannot be finalized. This example illustrates how crossing linkages can seriously affect the validity of a network level schedule.

3.5 Re-platforming shunting

Some train units arrive at a platform, finish all the necessary operations and later leave the station from the same platform without shunting away to any other place, for example, sidings or other platforms. However, some train unit blocks may leave from a different platform from its arrival platform, thus, re-platforming shunting must be operated. At the network level, station level shunting operations are not realized because of the ignorance of station layouts. In the UK railway industry, schedulers try to avoid re-platforming movements and accomplish the station operations for an arrival train unit at the same platform. When a re-platforming operation is inevitable, there are two essential constraints to be considered. One is if there is enough slack time for this re-platforming operation. The other is that the feasible time boundaries for re-platforming movements should be considered to ensure that there will be no blockage caused by this re-platforming operation at an inappropriate time. A given timetable defines the arrival time and departure time for logical trips. However, the departure times and arrival times for the re-platforming operations of the train units to be prepared to serve another trip at stations are flexible. The existing train units parking on the platforms may result in the fact that the involved re-platforming operations can only be operated in a specific time range. Also, re-platforming operations in different time ranges may result in different coupling orders.

3.6 Interchangeability between train units of the same type

A railway operator usually has many types of train units. The train units of the same type share common configurations, so that they are interchangeable. In a unit schedule, each train unit has a unique diagram with detailed routes and serving trips. When a train unit is not available, a substitute of the same type can replace its mission if the changed mission does not affect their necessary maintenance. This feature gives shunting operations at the station level great flexibility by swapping the diagrams of the same type of train unit. In this research, the coupling order of the train unit block consisting of the train units of the same type is considered as conflict-free because the potential blockages can be easily avoided by swapping the missions of these train units.

4 Model and formulation

4.1 DAG generation and decision variables

A DAG consists of a set of nodes and a set of directed arcs in which no cycle exists, denoted as $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. The node set $\mathcal{N}$ consists of the timetabled trips (N), and the source and the sink ($\{s,t\}$) that are added to represent siding/depot, i.e., $\mathcal{N} = N \cup \{s,t\}$. The arcs in $\mathcal{A}$ represent potential connections between any two nodes. The DAG is generated on a given timetable and a series of real-world requirements, e.g., basic turnaround time, O-D pairs of connected trips, permitted train unit types. The arcs in a DAG are classified into three types: (1) the arcs connecting an arrival to a departure at the same station are defined as trip-to-trip arcs (A); (2) the arcs between a trip and the source/sink are defined as sign-on (A_0)/sign-off (A_∞) arcs; (3) the arcs connecting an arrival to a departure at a different station are defined as empty running arcs (A_e). Thus, the arc set of the original DAG is $\mathcal{A} = A \cup A_0 \cup A_\infty \cup A_e$, where $A_0 = \{(0,i)|i \in N\}$, $A_\infty = \{(i,\infty)|i \in N\}$, $A = \{(i,j)|i,j \in N, \text{ and } s_a^i = s_d^j\}$, and $A_e = \{(i,j)|i,j \in N, \text{ and } s_a^i \neq s_d^j\}$. s_a^i and s_d^j represent the arrival location of trip i and the departure location of trip j. Let $\mathcal{K}$ denote the set of train unit types used in a given timetable. Based on the compatibility between train unit type and route, type graphs ($\mathcal{G}^k, \forall k \in \mathcal{K}$) are constructed based on the original DAG ($\mathcal{G}$). Each $\mathcal{G}^k$ is a subgraph of $\mathcal{G}$. A path $p \in \mathcal{P}^k$ represents a train unit diagram of type k on $\mathcal{G}^k$, which contains a collection of consecutive arcs and serving trips starting from the source and ending at the sink. Based on $\mathcal{G}$ and $\mathcal{G}^k$, decision variables are defined as follows:

- arc-type-flow variables: $x_a \in \mathbb{N}$, $\forall a \in \mathcal{A}^k$, $\forall k \in \mathcal{K}$, representing the number of train units of type k flowing on arc a.
- path-type-flow variables: $x_p \in \mathbb{N}$, $\forall p \in \mathcal{P}^k$, $\forall k \in \mathcal{K}$, representing the number of train units of type k flowing on path p.
- arc-selection (fixed charge) variables: $y_a \in \{0,1\}$, $\forall a \in \mathcal{A}$, indicating if an arc $a = (i,j)$ is selected in the solution.

The arc-type-flow and path-type-flow variables are inter-convertible by expression (2), where $\mathcal{P}_a$ represents all the paths containing type arc a.

$$x_a = \sum_{p \in \mathcal{P}_a} x_p, \forall a \in \mathcal{A}^k, \forall k \in \mathcal{K} \tag{2}$$

4.2 Mathematical model

Two main terms are considered in the objective function. The first term minimizes the total number of train units used to serve the entire timetable. The second term minimizes the general costs of operating the involved train units.

$$min\left(\sum_{k \in \mathcal{K}} \sum_{a \in \mathcal{A}_0^k} x_a + \sum_{k \in \mathcal{K}} \sum_{a \in \mathcal{A}^k} (c_a x_a)\right) \tag{3}$$

4.2.1 Constraints at the network level

Expression 4 represents the fleet size constraints (C_1). For each type $k \in \mathcal{K}$, there is an upper bound b_k. To schedulers, a lower bound (b'_k) can also be considered.

$$b'_k \leq \sum_{a \in \mathcal{A}_0^k} x_a \leq b_k, \ \forall k \in \mathcal{K} \tag{4}$$

Expression 5 demonstrates the flow conservation constraints (C_2) which are to ensure the flow of every train unit type on each trip node is balanced, where A_j^{in} and A_j^{out} are the get-in type arc set and get-out type arc set of trip j.

$$\sum_{a \in A_j^{in}} x_a - \sum_{a \in A_j^{out}} x_a = 0, \ \forall j \in N, \ \forall k \in \mathcal{K} \tag{5}$$

Each trip $i \in N$ has to satisfy three basic hard constraints: passenger demands (C_{3a}), unit coupling upper bound (C_{3b}) and carriage upper bound (C_{3b}), presented as expressions (6), (7), and (8), respectively. These constraints can be converted into equivalent convex hull constraints (C_3) that are tighter, as shown in expression 9. F_j is the convex hull facets for trip j. $H^j_{(f,k)}$ and h^j_f are coefficients at type k position and the corresponding RHS of facet f for trip j. More proofs can be found in Lin and Kwan (2016b).

$$\sum_{k \in \mathcal{K}_j} \sum_{a \in A_j^{in}} q^k x_a \geq q_j, \ \forall j \in N \tag{6}$$

$$\sum_{k \in \mathcal{K}_j} \sum_{a \in A_j^{in}} u^k x_a \leq u_j, \ \forall j \in N \tag{7}$$

$$\sum_{k\in\mathcal{K}_j}\sum_{a\in A_j^{in}} v^k x_a \le v_j,\ \forall j \in N \tag{8}$$

$$\sum_{k\in\mathcal{K}_j}\sum_{a\in A_j^{in}} H^j_{(f,k)} x_a \le h^j_f,\ \forall f \in F_j,\ \forall j \in N \tag{9}$$

Expression 10 is to ensure the consistency of the values between x_a and y_a (C_4). For each arc $a \in \mathcal{A}$, if it is selected in the solution: $y_a = 1$ and $x_a \ge 1$; otherwise: $y_a = 0$ and $x_a = 0$. Here, u_a is the biggest unit number that an arc can flow.

$$\sum_{k\in\mathcal{K}:a=a(k)} x_a \le u_a y_a,\ \forall a \in \mathcal{A} \tag{10}$$

4.2.2 Constraints at the station level

While considering the station level infrastructure, a selected arc is feasible if it is operable within permitted time and station layouts, and a train unit schedule is feasible if every arc in the schedule is a feasible arc. A conflict may occur when some arcs are selected in the same solution, defined as arc-selection conflict. To be more precise, a conflict is caused by the specific flow of certain types of train units on a set of arcs, defined as type-flow conflict. Thus, arc-selection constraints (C_{5a}) and type-flow constraints (C_{5b}) are modeled to eliminate station level conflicts (C_5).

An arc-selection conflict contains a set of arcs which together are not compatible at the station level, denoted by $\bar{A}$. If the arcs in $\bar{A}$ are not all selected in a solution, the specific conflict will not appear in the solution. Let $\mathcal{Z}_1$ be the set of possible arc-selection conflicts. A conflict-free schedule must not contain any arc set $\bar{A} \in \mathcal{Z}_1$. Expression 11 models the arc-selection constraints (C_{5a}).

$$\sum_{a\in\bar{A}} y_a \le \mid \bar{A} \mid -1,\ \forall \bar{A} \in \mathcal{Z}_1 \tag{11}$$

A type-flow conflict represents an infeasible set of arcs on which unit types and corresponding quantities are considered. Let $\mathcal{Z}_2$ represent the set of possible type-flow conflicts. These conflicts actually are integral points in the solution space. The integer cut technique proved by Balas and Jeroslow (1972) can be applied to eliminate infeasible binary integral points, represented as $p = (x_1, \ldots, x_i, \ldots x_n)$. The infeasible integral point is divided into two sets shown in expression (12). Since the variable x_i is binary, constraint (13) avoids a certain integral point.

$$B = \{i \mid x_i = 1\},\ Q = \{i \mid x_i = 0\} \tag{12}$$

$$\sum_{i\in B}(1 - x_i) + \sum_{i\in Q} x_i \ge 1 \tag{13}$$

This technique is only valid for binary variables. Necessary adaptation is derived below to apply it on an integer model. A set of working binary variables is introduced based on arc-type-flow variables, denoted as follows:

– arc-type-flow binary variables: $x_a^q \in \{0,1\}, \forall a \in \mathcal{A}^k, \forall k \in \mathcal{K}, \forall q \in \mathcal{U}$. $\mathcal{U}$ denotes the set of natural values which can be assigned to corresponding arc-type-flow integer variables.

For example, suppose the unit coupling upper bound is 3, i.e., $\mathcal{U} = \{0,1,2,3\}$, and type graph $\mathcal{G}^{k_1}$ is in use. For each arc-type-flow integer variable x_a, there would be four corresponding arc-type-flow binary variables: x_a^0, x_a^1, x_a^2, and x_a^3. The binary values for them are defined as expression (14)

$$x_a^q = \begin{cases} 1, \text{ if } x_a = q \text{ and } q \geq 1. \\ 0, \text{ otherwise.} \end{cases} \tag{14}$$

Let $\breve{A}$ represent the set of arcs containing a type-flow conflict; B contains all the type-flow binary variables with value 1, and Q holds the other variables over the arc set $\breve{A}$. Thus, the type-flow constraints (C_{5b}) are shown in expression (15), where $\mathcal{Z}_2$ is the set of $\breve{A}$.

$$\sum_B (1 - x_a^q) + \sum_Q x_a^q \geq 1, \ \forall \breve{A} \in \mathcal{Z}_2 \tag{15}$$

The domains for each type of variables are shown at the places where defined. There are some advantages and disadvantages of constraints C_{5a} and C_{5b}. C_{5a} are straightforward and easy to apply. However, these constraints are slightly over-tight and they might rule out some feasible solution because they do not consider any flow combination on the infeasible arc set. On the other hand, the type-flow constraints do not have that side effect and they are just tight enough to eliminate the infeasible type-flow conflict on the infeasible arc set. However, these constraints will largely increase the problem size since new variables need to be introduced.

The network level problem (constraints C_1 to C_4) has been researched during the last few years and many results have been achieved (Lin and Kwan 2016a; Copado-Mendez et al. 2017), defined as Phase I. The two alternative constraints of eliminating station level conflicts (shown in expressions (11) and (15)) are based on fixed-charge variables that largely increase the complexity of solving the model. Besides, computing infeasible arc sets for arc-selection conflicts and type-flow conflicts ($\mathcal{Z}_1$ and $\mathcal{Z}_2$) in advance as inputs based on the station level infrastructure and the original DAG is a massive work, because numerous of arc-selection/type-flow combinations are supposed to be checked. This makes solving the model as a whole almost impossible. On the other hand, the coupling orders for coupled train unit blocks are also left to be further determined. Since many of the station level constraints could have already been satisfied implicitly by the network level solution, an adaptive method is proposed to determine feasible coupling orders and resolve the station level conflicts based on the basic network level model, defined as Phase II.

5 Coupling order and network propagation boundaries

5.1 Coupling order definition

The solution from Phase I is a set of paths each corresponding to a certain train unit sequentially serving a set of trips. This solution gives a collection of units for each trip and connections among trips. There is no defined unit sequence in a coupled unit block because of the simplification of station level constraints. Hence, a multi-set can be defined to represent the collection of units for each trip, denoted by U_j, $j \in N$. The elements in U_j are the units of the same or different types serving trip j. The number of elements of the same unit type in U_j is the type flow of trip j. For instance $U_j = \{X, X, Y, Y\}$ means trip j is served by 4 units composed of 2 units of type X and 2 units of type Y. In a network level solution, two attributes for a trip j are important to introduce the concept of coupling order, which are the predecessor and successor node sets (I_j and R_j). The predecessor/successor node set of trip j contains all the nodes that have an arc to/from trip j. Considering Fig. 4 as an example, the predecessor node set of trip j is $\{i_1, i_2, \ldots, i_m\}$ and the successor node set of trip j is $\{r_1, r_2, \ldots, r_m\}$. In addition, the trip timing and direction information related to a given timetable and corresponding station structure are also important in later discussion. Their notations are shown in Table 1.

Here, $dp(j)$ and $ap(j)$ are binary values since each platform has two notional directions. Let us define them as $\{up, down\}$, which will be used in the rest of this part. One value has to be crossed out for the dead-end platforms since they only have one end accessible. Thus, the general unit-block collection for each trip can be described by expression (16). Normally, the number of units coupled together can be up to 4 in real-life, i.e., $m \leq 4$, $|I_j| \leq 4$, and $|R_j| \leq 4$.

$$U_j = \{u_1 u_2 \cdots u_m\}, \forall j \in N \tag{16}$$

For trips served by a single unit ($m = 1$), there is no need to consider their coupling order. Besides, the unit sequence of unit blocks of the same type is also not critical. This is based on the assumption that the diagram assignment among units of the same type can be easily swapped since unit circulation and maintenance are not considered in this research.

Day-time coupling and decoupling operations are mostly conducted at platforms in the UK. Those activities are critical because they form/request a certain unit

Table 1 Notations of some trip attributes

Notations ($j \in N$)	Definitions
I_j	Predecessor node set for trip j in a given solution
R_j	Successor node set for trip j in a given solution
$dp(j)$	Departing direction of trip j
$dt(j)$	Departing time of trip j at its departure platform
$ap(j)$	Arriving direction of trip j
$at(j)$	Arriving time of trip j at its arrival platform

sequence. This feature together with trip attributes in Table 1 can be utilized to tentatively assign locally feasible coupling orders to some trips. Let S_j denote the set of stopping stations of trip j. To formalize the concept of coupling order, let us define a sequenced multi-set O_j^s to represent the coupling order for trip j at location $s, s \in S_j, j \in N$. The start of a sequence is the front of a certain moving direction followed by sequenced unit blocks as the rear. During the coupling order assignment process, it has three possible states: fixed, unfixed, and semi-fixed, which can be described in a general expression (17). The elements in O_j^s are sequenced subsets of U_j. The union of all the elements in O_j^s must be the same with U_j, because O_j^s and U_j express the unit formation of trip j, as shown in expression (18). Generally, $O_j^s = U_j$ if O_j^s is "unfixed".

$$O_j^s = [v | v \subseteq U_j] = [v_1 v_2 \cdots v_{m'}], \forall s \in S_j, \forall j \in N \tag{17}$$

$$\bigcup_{i=1}^{m'} v_i = U_j, \quad \forall j \in N,\ m' \leq m \tag{18}$$

In the context of arrival and departure, a unit block has a front and a rear with respect to its travel direction. As shown in Fig. 3, the coupling of two unit blocks can either be a front-front (Fig. 3a) or front-rear (Fig. 3b, c) attachment.

Rear-rear (Fig. 3d) attachment is physically impossible unless one of the unit blocks reverses into the platform, in which case the reversed approaching end is regarded as the front. Similarly, after the decoupling of a unit block, the two resulting unit blocks may next travel in the same direction with front-rear facing or in the opposite directions with rear-rear facing (front-front facing is physically impossible).

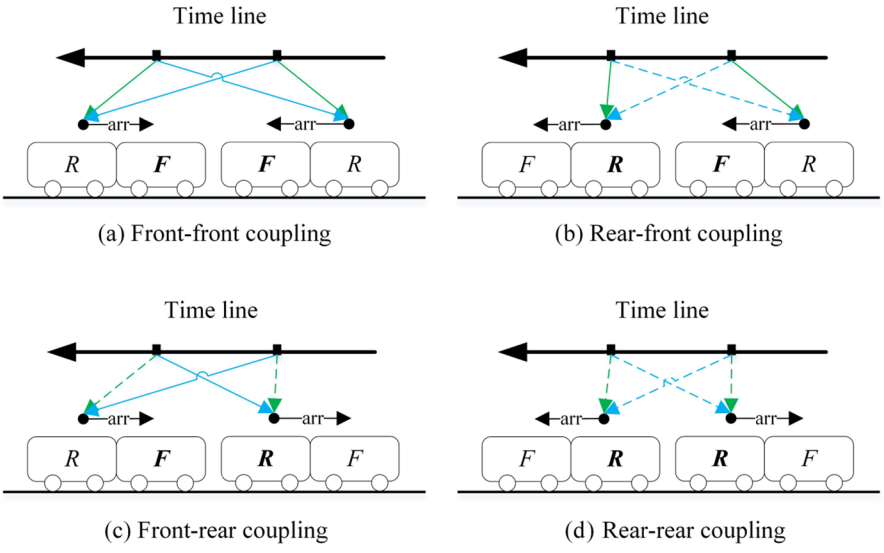

Fig. 3 Coupling position

5.2 Function and operator definitions

In real life, the moving direction of a trip may be reversed at some en-route stations. With a known unit sequence in a trip at a certain station, the unit sequences at other stations are all known by counting the number of en-route reversal operations. For a given network level schedule, it is worthy of tentatively seeking if there exists a feasible coupling order assignment. Coupling and decoupling operations can locally fix some feasible coupling orders. Running trips can spread the influence of a coupling order out to the whole network. Accordingly, a front-and-rear reversal function is defined to express the en-route reversal movement, which will be applied once the moving direction is reversed. In addition, coupling and decoupling operators are also defined based on the assumption that the trips involving coupling/decoupling events are operated at the same platform. Their notations are shown in Table 2. The operations related to different platforms, for example re-platforming, will be further explained in Sect. 6.1.3.

(1) Front-and-rear reversal function $Rev(\delta)$: The parameter δ is a coupling order. This function can be applied multiple times, defined as $Rev^n(\delta)$, $n \in \mathbb{N}$. While n is odd, δ will be reversed; otherwise, the same order is kept. Testing n is odd or even is a matter of computational implementation.

(2) Coupling operator: A coupling event is referring to two (sequenced) sets joining together to form a longer (sequenced) set. The arcs involving this operation are called coupling linkages. Let us define the coupling operator (+) on two unit blocks with the resultant coupling order as shown in expression (19). That is, the second operand unit block w is attached to the rear of the first operand unit block v.

$$u = v + w \Longrightarrow O_u = [O_v O_w] \tag{19}$$

Let i_1, i_2 denote train unit blocks arriving at the same platform to be attached and i_1 arrives first, $at(i_1) < at(i_2)$. Suppose i is the resultant unit block after the attachment. As shown in Fig. 3, i_1 and i_2 can arrive from the same/opposite direction(s). Considering $ap(i_1)$ as the *reference direction*, the coupling order of the resultant unit block is shown in expression (20).

Table 2 Function and operator

Symbols	Remarks
$Rev(\delta)$	Front-and-rear reversal function
"+"	Coupling operator
"−"	Decoupling operator

$$
\begin{aligned}
i &= \begin{cases} i_1 + i_2, & \text{if } ap(i_1) = ap(i_2) \\ Rev(i_2) + i_1, & \text{otherwise} \end{cases} \\
\Longrightarrow O_i &= \begin{cases} [O_{i_1} O_{i_2}] \\ [Rev(O_{i_2}) O_{i_1}] \end{cases}
\end{aligned}
\tag{20}
$$

Once the coupled formation O_i based on the reference direction is fixed, the next step is to assign O_i to trip j according to the relative directions between $dp(j)$ and $ap(i_1)$, seen in expression (21).

$$
O_j^{ori} = \begin{cases} O_i, & \text{if } dp(j) = ap(i_1) \\ Rev(O_i), & \text{otherwise} \end{cases} \tag{21}
$$

(3) Decoupling operator: Similar to the discussion on the coupling operator in (2), let us define the decoupling operator $(-)$ on two unit blocks with the resultant coupling order of the first operand as shown in expression (22), in which the second operand unit block w is detached from the rear of the first operand unit block v.

$$
u = v - w \Longrightarrow O_v = [O_u O_w] \tag{22}
$$

Let r, r_1, r_2 denote train unit blocks such that r_1 and r_2 depart from the same platform after being detached from r. Suppose r_1 departs first ($dt(r_1) < dt(r_2)$), and using $dp(r_1)$ as the *reference direction*. The coupling order of unit block r can be obtained according to expression (23).

$$
\begin{aligned}
r_1 &= \begin{cases} r - r_2, & \text{if } dp(r_1) = dp(r_2) \\ r - Rev(r_2), & \text{otherwise} \end{cases} \\
\Longrightarrow O_r &= \begin{cases} [O_{r_1} O_{r_2}] \\ [O_{r_1} Rev(O_{r_2})] \end{cases}
\end{aligned}
\tag{23}
$$

With the arrival direction of r, the requested coupling order at the destination of trip j can be assigned as expression (24).

$$
O_j^{dest} = \begin{cases} O_r, & \text{if } ap(j) = dp(r_1) \\ Rev(O_r), & \text{otherwise} \end{cases} \tag{24}
$$

For the dead-end platform, the unit blocks can only arrive from the same direction and the departure direction must be opposite of the arrival direction.

When there is no coupling ($|I_j| = 1$) or decoupling ($|R_j| = 1$) related to forming/decomposing some new unit blocks, the coupling order is temporarily considered as unconstrained. The coupling-order requirement of unfixed trips will be restored at the stage of coupling order propagation through the whole network. Another case shown in Fig. 4 illustrates that trips involve more than one coupling/decoupling operation, i.e., $|I_j| \geq 3$ or $|R_j| \geq 3$. This case is considered in an iterative way as shown in Algorithm 1. Only one coupling operation is applied at each iteration. The temporary coupling order during the iteration process is stored in O_{temp} till all of the unit block of trips in I_j is traversed. Thus the final O_i can be

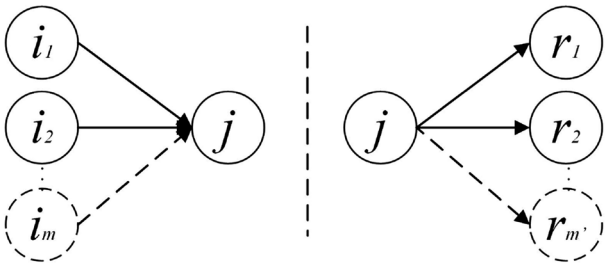

Fig. 4 Multiple (de-)coupling operations

obtained to be assigned to O_j^{ori}. Similarly, this method can be applied to multi-decoupling operations as well.

Algorithm 1 Multi-coupling operations

Require: I_j
Ensure: O_j^{ori}
1: $I_j^{'} := sorted(I_j)$; $i_1 := I_j^{'}.firstTrip$; $O_{temp} := O_{i_1}^{dest}$
2: **for all** idx **in** $I_j^{'} \setminus i_1$ **do**
3: $O_{temp} \leftarrow$ "+" for O_{temp} and $O_{i_{idx}}$
4: **end for**
5: $O_i := O_{temp}$
6: **if** $dp(j) = ap(i_1)$ **then**
7: $O_j^{ori} := O_i$
8: **else**
9: $O_j^{ori} := Rev(O_i)$
10: **end if**

5.3 Coupling order propagation boundaries

In this section, we systematically extract concise parts of the DAG where coupling-order conflicts could arise in a given Phase I solution because the coupling order has not been maintained at the network level. With respect to the full DAG $\mathcal{G}$, a Phase I solution is a subgraph $\mathcal{G}^* \subset \mathcal{G}$ with train unit flows assigned. Disregarding the source and sink, $\mathcal{G}^*$ is decomposed into one or more disjoint connected graphs, which can be classified into two types as follows, represented as sets $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively.

- We do not consider the coupling order issue for a subgraph if it has only a single unit type or it has multiple unit types but does not involve any coupling/decoupling operation, stored in $\mathcal{G}_1$.
- The subgraphs which do not satisfy the conditions for $\mathcal{G}_1$ will be stored in $\mathcal{G}_2$ because their coupling order issues matter.

In a $g \in \mathcal{G}_2$, there may be some trips that are served by only one single unit type, denoted as set N_1. The coupling order issues for those trips are trivial, thus, g can be further decomposed to smaller subgraphs to analyze the coupling order issues by trimming off all the trips in N_1. Let us use $\mathcal{G}_3$ to denote the set of subgraphs decomposed from all the graphs in $\mathcal{G}_2$.

Figure 5, shows an example graph $g \in \mathcal{G}_2$ in which the trips that are free of coupling order issue (single unit type) are represented as dashed circles. Two independent subgraphs (marked out by blue boxes) can be extracted from g when those dashed nodes are trimmed off. A graph in $\mathcal{G}_3$ is the smallest unit to analyze the coupling order issue on the network. The coupling order issue of a $g_1 \in \mathcal{G}_3$ has nothing to do with that of another $g_2 \in \mathcal{G}_3$, but is sealed within g_1 such that the coupling orders for the trips contained in g_1 interact with each other, defined as coupling order propagation. If the coupling orders of the trips in a graph $g_1 \in \mathcal{G}_3$ are not compatible with each other, all the get-in and get-out arcs of the nodes in g_1 will be collected as an infeasible arc set. Take the second subgraph in Fig. 5 as an example. If the coupling orders of T14, T15, T17 are not compatible with each other, arcs {T11–T14, T10–T14, T14–T16, T15–T16, T16–T17, T16–T18} will be collected as an infeasible arc set.

To illustrate how a coupling order propagates through a graph in $\mathcal{G}_3$, the first subgraph in Fig. 5 is taken as an example. At its fringe, all the get-in and get-out arcs and their connected nodes (T3, T4, T9, T10, and T12) are restored, as shown in Fig. 6 with one coupling and two decoupling operations. One en-route reversal operation (marked as *) will happen to T3, T8, T10 and T12.

Figure 6b is the corresponding schematic space-time diagram and their serving units assigned at the network level, on which (de-)coupling operations are marked as black circles. Stations A, B, C have only one platform and their platform types are also indicated. There may be other trains visiting as intermediate stops, which are not shown here. They are usually planned at the timetabling stage such that they will not be in conflict with other terminating trips.

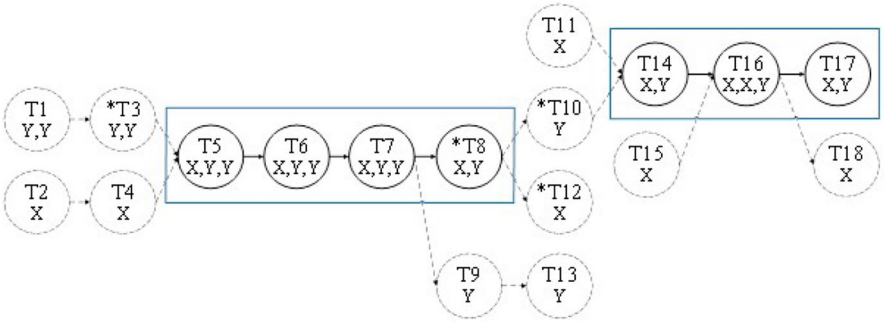

Fig. 5 Two subgraphs extracted from a graph in $\mathcal{G}_2$

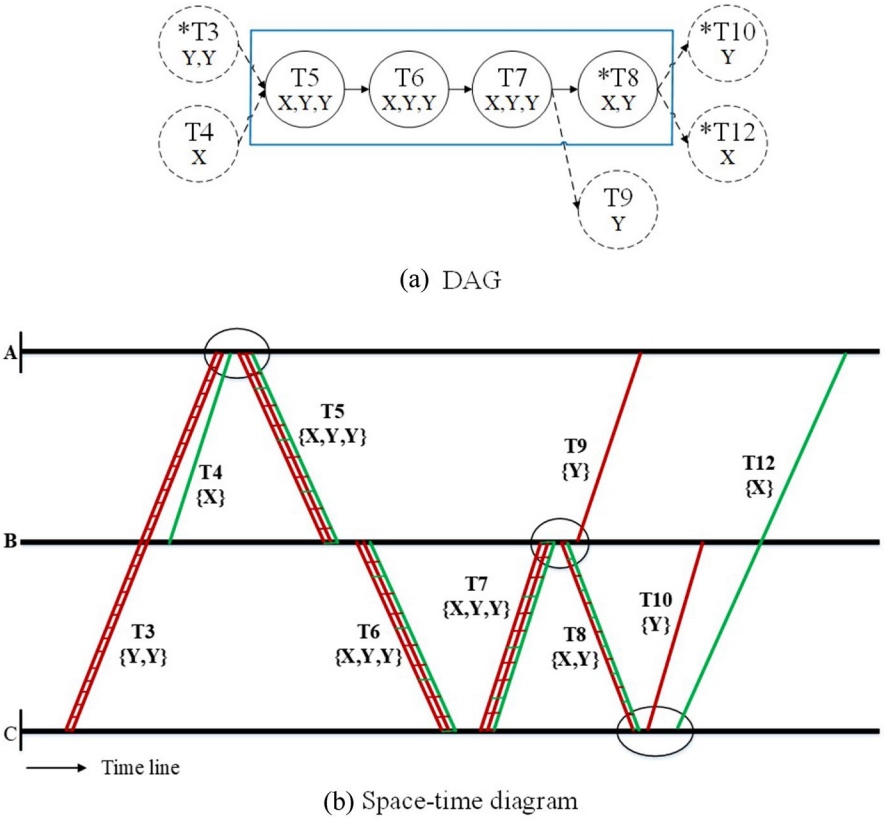

Fig. 6 A subgraph extracted from Fig. 5

First of all, the algorithm assigns local feasible coupling orders by applying the coupling/decoupling operations introduced in Sect. 5.2 and the results are as follows:

(1) Coupling operation to form T5 with regard to arcs (T3,T5) and (T4,T5) by considering $ap(T3)$ as the reference direction.
Since, $ap(T3) = ap(T4)$ and $at(T3) < at(T4)$
We have, $i = i_3 + i_4 \Longrightarrow O_i = [O_3 O_4] = [\{Y, Y\}\{X\}] = [YYX]$
Since $dp(T5) \neq ap(T3)$, $O_5^{ori} = Rev(O_i) = [XYY]$
Thus, $O_5^{dest} = O_5^{ori} = [XYY]$
Since, no en-route reversal operation for T5, $O_5^{ori} = O_5^{dest} = [XYY]$.

(2) Decoupling operation for T7 through arcs (T7,T8) and (T7,T9) to serve T8 and T9 with consideration $dp(T8)$ as the reference direction.
As $dp(T8) \neq dp(T9)$, $r_8 = r' - Rev(r_9)$
Thus, $O_{r'} = [O_8 Rev(O_9)] = [\{X, Y\}\{Y\}] = [\{X, Y\}Y]$
Since $ap(T7) = dp(T8)$, $O_7^{dest} = O_{r'} = [\{X, Y\}Y]$
Then, $O_7^{ori} = O_7^{dest} = [\{X, Y\}Y]$

(3) Decoupling operation for T8 to serve T10 and T12 over arcs (T8, T10) and (T8, T12). Let $dp(10)$ be the reference direction.
Since, $dp(T10) = dp(T12)$ and $dt(T10) < dt(T12)$
We get, $r_{10} = r - r_{12} \Longrightarrow O_r = [O_{10}O_{12}] = [\{Y\}\{X\}] = [YX]$
Because of $ap(T8) \neq dp(T10)$, $O_8^{dest} = Rev(O_r) = [XY]$
Thus, $O_8^{ori} = Rev(O_8^{dest}) = [YX]$

Through the operations above, the coupling orders of T5 and T8 are fixed and the coupling order of T7 is semi-fixed, but the coupling order of T6 is still unfixed. Three possible tracking methods are considered for the coupling order propagation: early-to-late tracking, late-to-early tracking, and tracking starting with any intermediate trip. These methods may result in conflict-free or coupling-order collision at different locations on the network.
Method 1: T5 → T6 → T7 → T8.

- T5 → T6: Since there is no en-route reversal happening to T6 and $ap(T5) = dp(T6)$, thus, $O_6^{ori} = O_6^{dest} = O_5^{dest} = [XYY]$.
- T6 → T7: Because $ap(T6) \neq dp(T6)$ and no en-route reversal for T7 as well, thus, $O_7^{ori} = O_7^{dest} = Rev(O_6^{dest}) = [YYX]$.
- Compare the propagated coupling order to the semi-fixed coupling order requested by the decoupling operation: $[YYX]$ is not compatible with $[\{X, Y\}Y]$.
- Conclusion: The coupling-order collision is at T7.

Method 2: T8 → T7 → T6 → T5.

- T8 → T7: This involves a decoupling operation which is considered at the stage of partially finalizing the locally feasible coupling order of T7. With a known coupling order of T8, $O_7^{dest} = [O_8^{ori} Rev(O_9^{ori})] = [YXY]$. This coupling order must be compatible with the semi-fixed coupling order of T7 because this propagation only uses the fixed coupling order of the unit block serving T8 to replace the unfixed part in $[\{X, Y\}Y]$.
- T7 → T6: $O_6^{ori} = O_6^{dest} = Rev(O_7^{ori}) = [YXY]$.
- T6 → T5: $O_5^{ori} = O_5^{dest} = O_6^{ori} = [YXY]$.
- Compare the propagated coupling order to the fixed coupling order formed by the coupling operation: $[YXY]$ is not compatible with $[XYY]$.
- Conclusion: The coupling-order collision is at T5.

Method 3: Tracking starts with any intermediate unfixed trip, T6 in this example. The coupling order of T6 can be fixed via T5 or T7.

- T5 → T6: $O_6^{ori} = O_5^{dest} = [XYY]$.
- T7 → T6: $O_6^{dest} = Rev(O_7^{ori}) = [Y\{X, Y\}]$.
- O_6^{ori} and O_6^{dest} are not compatible with each other.
- Conclusion: The coupling-order collision is at T6.

As shown in the example above, although the coupling-order collision might be located in different places because of different propagation tracking methods, the same sets of arcs are involved, which are all the get-in and get-out arcs of the nodes in a graph $g \in \mathcal{G}_3$. For a given network level schedule, it may result in some individual coupling-order collisions, and it is also possible that there are some collisions with overlapping arcs. For the latter case, the collisions with overlapping arcs can be considered via two strategies: considering them individually such that each conflict contains the overlapping arcs, or grouping them together as a combined conflict. Some experiments on these strategies will be reported in Sect. 7.2.

6 An adaptive approach

Since many of the station level constraints could have already been satisfied implicitly by the basic network level solution in Phase I, this paper proposes an adaptive method to resolve the station level constraints based on the basic network model. Figure 7 shows a flowchart of this method. The basic Phase I containing constraints C_1 to C_4 (referring back to Sect. 4.2.1) is solved by *RS-opt* in Lin and Kwan (2016a). Phase II attempts to detect potential conflicts and assigning feasible coupling orders. Once an unresolvable station level conflict is encountered, a corresponding station level constraint C_5 (seen in Sect. 4.2.2) will be added to *RS-Opt*. A new solution is to be sought once new station level constraints are added. If no more station level blockage is detected, the corresponding feasible coupling orders of trips served by multi-units will also be given such that a solution with richer coupling order information will be delivered.

Phase II contains two core parts. The first part is the conflict detection and coupling order assignment based on a series of factors, for instance, physical structure of railways and stations, timing and moving directions of trips, tentatively assigned

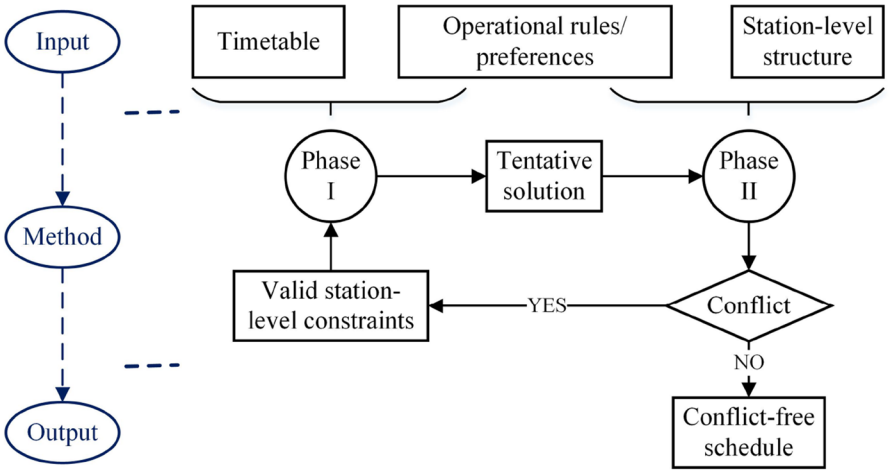

Fig. 7 Flowchart for the adaptive approach

arcs and unit flows for trips, etc. For a given station operation environment, conflicts are detected when a tentative linkage is not operable, or feasible coupling orders cannot be found at a certain platform, or a preassigned coupling order is not feasible to another operation environment. The second part is to extend the model of *RS-Opt* and resolve detected conflicts.

6.1 Coupling order assignment and conflict detection

A two-stage method is proposed to assign coupling orders and detect potential blockages caused by crossing linkages or coupling-order collision. The first stage is based on each platform which has two purposes: one is to assign locally feasible coupling orders with regard to coupling/decoupling operations introduced in Sect. 5.2; the other is to verify related linkage feasibility. The second stage is to find out the compatibility among the locally fixed coupling order within each $g \in \mathcal{G}_3$ and also further fix some unfixed/semi-fixed coupling orders by spreading locally feasible coupling orders assigned by the first stage to the network.

6.1.1 Platform-based stage

This stage mainly assigns locally feasible coupling orders and detects potential conflicts caused by crossing linkages. The platform function can be modeled as a data structure behaving similar to a double-ended queue for storing and processing unit blocks, in which unit blocks are sequenced by "push" and "pop" operations, shown in Fig. 8. A "push" operation refers to one unit block getting into the platform with a "journey" and a "pop" operation refers to one unit block getting out of the platform with a "journey" after some necessary operations.

The "journey" here can be either a timetabled trip or a shunting movement, for example a re-platforming movement. The two ends for "push/pop" operations correspond to two approaching directions to a through platform, which can be mapped to *up* and *down* directions. The dead-end platform can also be modeled by disabling one end. Besides, this data structure has a capacity limitation regarding to the platform length. While pushing each arrival unit block, virtual "dividers" are added to isolate it from the existing unit blocks since unit blocks are physically separate when they arrive. This data structure is denoted as *unitStore*.

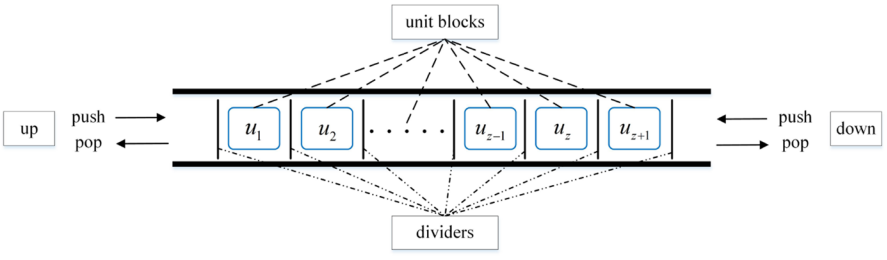

Fig. 8 Data structure for *unitStore*

Algorithm 2 describes the process at a platform. It takes a platform and basic solution s of Phase I as input. The output is *a richer train unit schedule* (hs') with locally feasible coupling orders assigned at each platform and verified linkages. Conflicts will be collected in CF_1 to be further resolved.

Algorithm 2 Platform-based stage

Require: $s,\ h, (\forall h \in \mathcal{H})$
Ensure: s' *and* CF_1
1: $s' := s$; $dL := sortedDepTripList$; $aL := sortedArrTripList$; $dtrip := dL.firstTrip()$; $time := dtrip.depTime$; $unitStore := empty$
2: **repeat**
3: **for all** $atrip$ **in** $A_{dt} = (aL \mid atrip.arrTime < time)$ **do**
4: **if** $(unitStore.length + atrip.length \leq max)$ **then**
5: $unitStore.push(atrip.composition)$
6: **else if** $(\exists i \in unitStore$: $shuntAway(i)$ **and** $i.length \geq atrip.length)$ **then**
7: $sh := shuntingMove$; $s'.update(sh)$; $unitStore.remove(i)$;
8: $unitStore.push(atrip.composition)$
9: **else**
10: $dt := \{unitStore.depTrips, link(atrip).headNodes\}$
11: $c := OCconflict$; $CF_1.add(c)$. // Over capacity conflict
12: $dL := dL.remove(dt)$; $aL := aL.remove(atrip)$; $unitStore.clear()$
13: **if** $dL.isEmpty() = false$ **then**
14: $dtrip := dL.firstTrip()$; $time := dtrip.depTime$; **go to** for loop
15: **else**
16: **break;**
17: **end if**
18: **end if**
19: $aL.remove(atrip)$
20: **end for**
21: $links := dtrip.getInLinkages$
22: $verify := linkImplement(links, unitStore)$
23: **if** $verify.operable = true$ **then**
24: $orders := verify.orderList$
25: $s'.update(orders)$; $unitStore.pop(dtrip.composition)$
26: **else**
27: $c := verify.LIconflict$
28: $CF_1.add(c)$. // Linkage implementation conflict
29: $u_c := c.linkedUnitblocks$; $unitStore.remove(u_c)$
30: **update** *element sequence in unitStore*
31: **end if**
32: $dL.remove(dtrip)$; $dtrip := dL.firstTrip()$; $time := dtrip.depTime$
33: **until** $dL.isEmpty()$

Three lists are formed for platform h: arrival list, departure list and linkage list connecting the arrivals to departures. This algorithm starts with the following initializations: *dL* and *aL* are time-sorted departure and arrival trips, respectively; *dtrip* and *time* are assigned with the first departure trip and its departure time, respectively; *unitStore* is initialized as empty. Once *time* is assigned a new value, all the trips in *aL* whose arrival times are smaller than *time*, defined as set A_{dt}, will be attempted to push in *unitStore* once there is enough space. If there is not enough space, the *shuntingAway*() function will be called to verify all the existing unit blocks to find out if some of them can be shunted away to secure

enough space for the new push-in unit block. The linkage slack time will be verified during this process. If *shuntingAway*(*i*) = *true*, the new push-in unit block can be stored in *unitStore* and accordingly the shunting movement will be saved in s'. Otherwise, all the arcs linked with those unit blocks will be considered as a conflict stored in CF_1 because they invalidate the platform capacity constraint, denoted as *OCconflict*. All the unit blocks will be cleared out from *unitStore*, and the departure trips corresponding to the unit blocks in *unitStore* and *atrip* will be removed from *dL*, and new *dtrip* and *time* will be assigned to restart the for loop.

After all the arrival unit blocks arriving earlier than the current *time* are successfully pushed into *unitStore*, the algorithm starts to deal with the unit blocks which are supposed to be popped out to serve the trip whose departure time is *time*. A function called *linkImplement*(), shown in Algorithm 3, is applied to verify the feasibility of *links* which are the get-in linkages of this departure trip with respect to the current *unitStore* status. If all the get-in linkages are verified as operable, the locally feasible coupling orders will be updated to s' and the unit block serving this trip will be popped out of *unitStore*. Otherwise, the conflict related to the linkage implementation, denoted as *LIconflict* will be saved into CF_1 to be resolved later and the unit blocks related to this conflict u_c will be removed from *unitStore*. Then *dtrip* and *time* will be renewed for the next iteration. A conflict will not break this algorithm until all the departure trips are verified because this strategy tries to collect as many conflicts as it can.

Algorithm 3 $linkImplement()$

Require: $links$, $unitStore$
Ensure: $orderList$, $LIconflict$, $operable$
1: $c := links.count()$; $ubs := unitStore(dir, c)$
2: $i := ubs.firstArrTrip$; $j := ubs.lastArrTrip$; $operable = true$
3: **if** $links.tailTrips = ubs.arrTrips$ **then**
4: **if** $c = 1$ **and** $i.getOutArc.count() \geq 2$ **then**
5: $orderList.add(O_i^{dest})$. // Assigned by " − ", case ②
6: **else if** $c \geq 2$ **then**
7: $counter := 0$
8: **for all** at **in** $ubs.arrTrips$ **do**
9: **if** $at.getOutArc.count() = 1$ **then**
10: $counter = counter + 1$
11: **end if**
12: **end for**
13: **if** $counter = c$ **then**
14: $orderList.add(O_{dtrip}^{ori})$. // Assigned by " + ", case ③
15: **else if** $counter = c - 1$ **and** $j.getOutArc.count() \geq 2$ **then**
16: $orderList.add(O_{dtrip}^{ori}, O_j^{dest})$. // Assigned by both " + " and " − ", case ④
17: **else**
18: $operable = false$; $LIconflict$. // No feasible order assigned
19: **end if**
20: **end if**
21: **else**
22: $operable = false$; $LIconflict$. // Crossing linkages
23: **end if**

Algorithm 3, *linkImplement*(), is a sub-part of Algorithm 2 and the coupling and decoupling operations ("+", "−") defined in Sect. 5.2 are applied in this algorithm. There are two types of input for this algorithm: one the the *links* obtained at line 19 in Algorithm 2, the other is the current status of *unitStore*. This algorithm focuses on verifying if the linkages in *links* of a departure trip (*dtrip*) are operable under the corresponding *unitStore* circumstances and returns an assigned locally feasible coupling order or a conflict. Let us define *dir* as the end (*up* or *down*) of *unitStore* that complys with the departure direction of *dtrip* and c is the total number of linkages in *links*. Thus, we define a function *unitStore*(*dir*, c) to obtain c unit blocks counted from the end *dir*, stored in *ubs*. i and j are the arrival trips delivering the first and last unit blocks in *ubs*, respectively. Let us consider an example by using the *unitStore* status shown in Fig. 8. Suppose $c = 3$, $dir = down$, thus, $ubs = [u_{z+1}u_zu_{z-1}]$, and i and j refers to the arrival trips delivering u_{z+1} and u_{z-1}, respectively. This algorithm firstly justifies if the unit blocks in *ubs* are delivered by the arrival trips which are actually linked to *dtrip*. If this condition is satisfied, the operations converting the linked arrival trips to *dtrip* will be investigated, which can be classified into four cases: ① no coupling or decoupling, ② decoupling only, ③ coupling only, ④ both coupling and decoupling, as shown in Fig. 9.

For case ① it will be claimed as operable if this condition is satisfied because it only requests standard turnaround time that has been considered while generating the DAG. The cases ② ③ ④ can be differentiated by counting c and get-out arcs of each arrival trip for the unit blocks in *ubs*. The feasibility of these cases will be ensured if a locally feasible coupling order can be assigned. Otherwise, this algorithm claims the given *links* as inoperable and returns a conflict.

6.1.2 Network-based stage

The result from the platform-based stage is the basis of the network-based coupling order propagation process which aims to finalize more feasible coupling orders or to detect conflicts caused by incompatible coupling orders within a graph in $\mathcal{G}_3$. As station operations are connected by running trips, some unfixed/semi-fixed trips can be further determined taking advantage of the coupling order propagation on the network based on the solution graph of Phase I and the infrastructure of railway networks, and the coupling order conflict can also be captured during the propagation process. Issues considered at this stage include: coupling order propagation, en-route reversal of a unit block, and flexible timings for some empty shunting movements (discussed in Sect. 6.1.3).

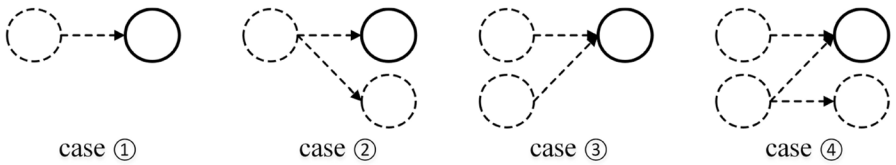

Fig. 9 Four possible cases of operations to form a departure trip

Algorithm 4 Network-based stage

Require: s', θ
Ensure: s'' *and* CF_2
1: $\mathcal{G}_3 := graphSplit(s')$; $s'' := s'$
2: **for all** g **in** $\mathcal{G}_3$ **do**
3: $i := g.fixedL.anyTrip$
4: $backwardSearch(eL_i)$; $forwardSearch(lL_i)$
5: **if** $propatingToEdge = true$ **then**
6: $orders := finalizedOrder(g)$
7: $s'' := s'.update(orders)$
8: **else**
9: $c := COconflict$; $CF_2.add(c)$. // Coupling-order collision in g
10: **end if**
11: **end for**

Algorithm 4 focuses on coupling order propagation through each $g \in \mathcal{G}_3$ that may contain some trips with fixed coupling orders assigned at the platform stage. Let us define *fixedL* as the fixed-order trip list for a $g \in \mathcal{G}_3$. For a trip $i \in$ *fixedL*, considering i as a divider, the trips within g can be split into two sorted lists. One includes the trips earlier than i, called *earlier trip list* (eL_i); the other contains the trips later than i, called *later trip list* (lL_i). Starting from i, two directions coupling order propagation search will be launched to further assign coupling orders and detect coupling-order collisions, as shown in Fig. 10. Trip i can be the first trip, the last trip, or any other trip, corresponding to one of those three different tracking methods of the coupling order propagation that have been analyzed in Sect. 5.3.

The *backwardSearch*(eL_i) and *forwardSearch*(lL_i) start with either O_i^{ori} or O_i^{dest}. It is possible that the platform stage assigns two incompatible orders to the origin and destination of trip i such that this g is claimed as a coupling-order conflict directly. Otherwise, the coupling order of trip i at a certain location will be propagated to other trips in g and compared to their coupling orders that may have been assigned at the platform stage. If the propagated order and the assigned order of each trip in g are compatible, g is claimed as conflict-free. The coupling orders of trips in g are finalized and updated to s''. Otherwise, all the linkages related to the trips in g will be collected as a coupling-order collision to be stored in CF_2 for further resolution.

6.1.3 Flexible timings

The time available for unit blocks moving from arrivals to departures is limited, which is rigidly constrained by the timetable. Within this limited time range, a series of operations must be accomplished at the corresponding stations such as coupling, decoupling, re-platforming, shunting to depot/siding, cleaning, equipment inspection, etc. Suppose that a unit block u arrives with trip i at platform h and finishes all

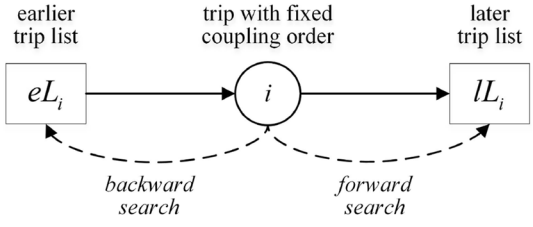

Fig. 10 Backward and forward search within a $g \in \mathcal{G}_3$

necessary operations which consume time of tp_h and later leaves from h to another platform h' to serve trip j. A dummy trip $\tilde{u}_{(i,j)}$ from h to h' is generated and its departure time $dt(\tilde{u})$ and arrival time $at(\tilde{u})$ are flexible. However, the duration time of dummy trip u is restricted by the clock time of the given timetable such that time boundaries for the departure and arrival times of dummy trips must be considered to ensure no blockage is caused at an inappropriate time. Normally, $\tilde{u}$ can be moved away from h after the necessary operations are finished but it must be shunted away before the next arriving trip, and its departure time range is shown in expression (25). Besides, $\tilde{u}$ must arrive at h' earlier than the departure time of trip j minus necessary departure operations and later than the last departure trip at h', and its arrival time range is shown in expression (26).

$$at(i) + pt_h < dt(\tilde{u}) < at(i+1),\ \forall \tilde{u} \tag{25}$$

$$dt(j-1) < at(\tilde{u}) < dt(j) - pt_{h'},\ \forall \tilde{u} \tag{26}$$

Figure 11 shows a simple example of avoiding blockage by manipulating flexible time boundaries together with the coupling order decision. Suppose that h_1 and h_4 are dead-end platforms and h_3 is a through platform and the directions of $T3$ and $T4$ are the same. $T1$ is a re-platforming trip from h_2 to h_1. The following two procedures explain which time boundary is feasible.

Procedure 1. $at(T2) < at(\tilde{u}_1) < dt(T3) - tp_{h_1} \rightarrow O_3^{ori} = [XY] \rightarrow O_3^{dest} = Rev(O_3^{ori}) = [YX] \rightarrow O_4^{ori} = [YX] \rightarrow O_4^{dest} = Rev^2(O_4^{ori}) = [YX] \rightarrow u_1$ blocks the departure of $u_2 \rightarrow$ infeasible time boundaries.

Procedure 2. $at(\tilde{u}_1) < at(T2) \rightarrow O_3^{ori} = [YX] \rightarrow O_3^{dest} = Rev(O_3^{ori}) = [XY] \rightarrow O_4^{ori} = [XY] \rightarrow O_4^{dest} = Rev^2(O_4^{ori}) = [XY] \rightarrow O_4^{ori} = [Y]$ and $O_4^{ori} = [X] \rightarrow$ feasible time boundaries.

6.2 Conflicts resolving

The station level resolution includes two main parts: coupling order assignment and conflict detection, and conflicts resolving. The first part is important because it is the basis for the second part. If the first part claims a conflict-free solution, there is no need to launch the second part anymore. These two parts are complementary to each other. After conflicts are detected by the first part, the second part converts these

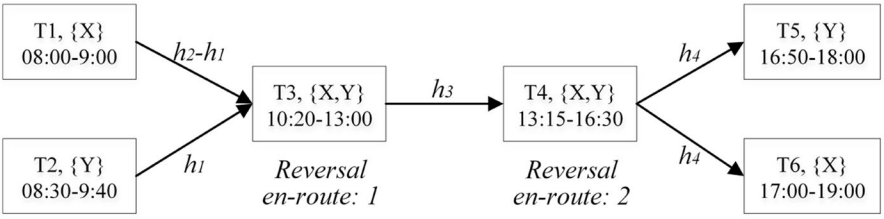

Fig. 11 Avoid blockage by manipulating the flexible timings boundaries

conflicts into linear constraints and feeds new constraints to *RS-Opt*. The method to form new constraints of eliminating the solutions that contain conflicts is described in Sect. 4.2.2. However, generating the full conflict set of $\mathcal{Z}_1$ or $\mathcal{Z}_2$ in an original full DAG is complicated as numerous arc/flow combinations exist. Through the proposed adaptive approach, the complication of generating $\mathcal{Z}_1$ or $\mathcal{Z}_2$ is avoided, and we only need to detect the conflicts existing in a Phase I solution. According to the observation on the Phase I solution, potential conflicts at the station level are sparse, thus, the difficulty of automatically assigning coupling orders and detecting potential conflicts is much lower than that of generating the full conflict set $\mathcal{Z}_1$ or $\mathcal{Z}_2$ in an original DAG. The method of assigning feasible coupling orders and detecting conflicts have been systematically designed in Sect. 6.1. Given a solution from Phase I, the detected conflicts are stored in CF_1 and CF_2 such that the constraints introduced in expressions (11) and (15) can be converted as expressions (27) and (28), defined as valid cuts.

$$\sum_{a \in \bar{A}} y_a \leq \mid \bar{A} \mid -1, \ \forall \bar{A} \in CF_1 \cup CF_2 \tag{27}$$

$$\sum_{B} (1 - x_a^q) + \sum_{Q} x_a^q \geq 1, \ \forall \breve{A} \in CF_1 \cup CF_2 \tag{28}$$

Figure 12 shows the logic of feeding valid cuts to the solver (*RS-Opt*) of Phase I. *RS-Opt* considers the original DAG $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ as input and gives a tentative solution $\mathcal{G}^* = (\mathcal{N}, \mathcal{A}^*)$. The station level attempts to assign feasible coupling orders to $\mathcal{G}^*$ and finalizes tentative linkages. During this process, the conflicts are detected at the platform stage and the network stage. Note that some conflicts at the platform stage can be locally resolved, which will not be recorded in CF_2. Three strategies are introduced to resolve local conflicts, including the swapping part of the unit diagrams for the same type of train unit, inserting extra station shunting movements within the bearable time of corresponding linkages, and manipulating the flexible timing of re-platforming/depot-return train units. The resolved platform-based results will be passed to the network-based stage to assign further coupling orders and detect coupling-order conflicts stored in CF_2. The conflicts in CF_1 and CF_2 are formed as valid cuts to be added to *RS-Opt*. *RS-Opt* will be launched again with detected valid cuts to deliver a new solution $\mathcal{G}^* = (\mathcal{N}, \mathcal{A}^*)$. The working mechanism of the valid cuts is to eliminate all the solutions containing the conflicts formed in the valid cuts. Thus, *RS-Opt* with added valid cuts will not consider a solution which contains any conflict that has been constrained in *RS-Opt* as a feasible solution. For a given schedule, if and only if no conflicts or only local resolvable conflicts are detected, the schedule will be reported as the final conflict-free schedule with assigned coupling orders and finalized linkages.

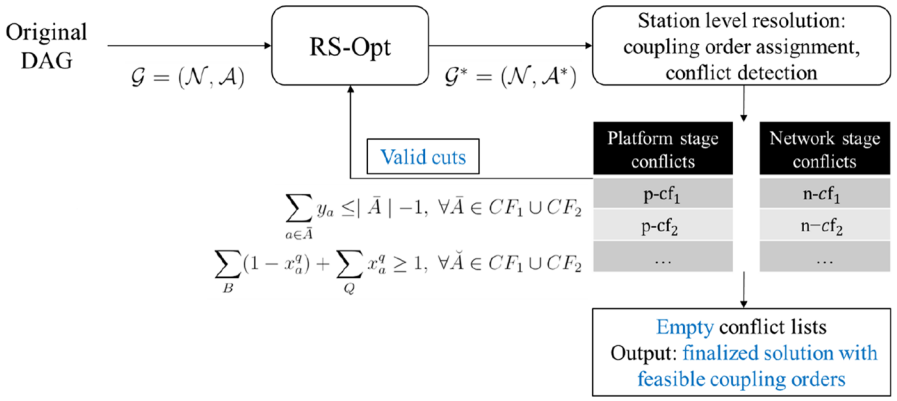

Fig. 12 Logic of feeding valid cuts to Phase I

7 Computational experiments

7.1 Dataset description

To verify the correctness of the concepts and approach proposed in this research, experiments were firstly conducted based on a small size artificial station level structure, shown in Fig. 13. This structure contains three terminating stations S1, S4, and S8, and their platform types are indicated.

The other stations are intermediate stations where the coupling order reversal en-route may occur. {*up*} and {*down*} directions for each platform are marked to notify the relative approaching and leaving directions of unit blocks. Only one direction is available for a dead-end platform, but two directions are usable for a through platform. One depot connects with the *down* direction of the two platforms of station S4, which means the units shunt to depot at this station can only leave their platforms through the *down* direction. Based on this station level infrastructure, four artificial datasets, containing trips running between all three O–D pairs (S1–S4, S1–S8, S4–S8). These datasets are small but well designed to cover the issues described in this research. Table 3 gives a summary of these artificial datasets.

Datasets D3 and D4 have the same timetable such that they have the same number of generated arcs. However, some trips in this timetable have different passenger demands, thus, their solutions could be different as well. Two types of compatible units are used on the network to satisfy diverse passenger demands.

The real-world datasets used in this research are derived from a British train operating company, TransPennine Express (TPE), which runs the regional and intercity rail

Fig. 13 Station level structure

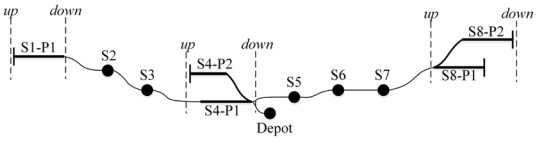

Table 3 Summary of artificial data sets

Dataset	DAG	Unit types	Unit compatibility	Fleet size
D_1	(17, 73)	X,Y	Yes	2X,4Y
D_2	(25, 111)	X,Y	Yes	2X,5Y
D_3	(21, 91)	X,Y	Yes	5X,5Y
D_4	(21, 91)	X,Y	Yes	5X,5Y

Table 4 Summary of real-world data sets

Dataset	DAG	Unit types	Unit compatibility	Fleet size	Stations
*Sub*1	(97, 2671)	c185, c350	Yes	20, 21	9
*Sub*2	(126, 1913)	c185, c350	Yes	11, 17	7
*Sub*3	(65, 569)	c185, c350	Yes	6, 12	4

services between the major cities of Northern England and Scotland. Usually, TPE has a daily timetable of around 500 trips. *RS-Opt* is set as the full model containing the arc-selection variables (y_a), which struggles to deliver a solution for even small instances (Lin and Kwan 2017). Thus, three sub-datasets are derived based on locations from the complete timetable as shown in Table 4. Class 185 and Class 350 are usually not coupled in real-life scheduling, but this research considers them to be able to couple with other units to test the coupling order issues.

The experiments were applied by the station level constraints C_{5a} making use of the fixed-charge variables (y_a) because C_{5b} is hard to be implemented in the existing *RS-Opt*. *RS-Opt* is written in FICO Xpress-MP 8.5 with Mosel, and the coupling order assignment and linkage finalization are coded in C#. The experiments are conducted on a 64-bit workstation with 64G and an Intel Core i7-6700HQ CPU.

7.2 Results of artificial datasets

Dataset D_1 generated a solution without any station level constraints added; it is conflict-free and feasible coupling orders can be assigned. The platform-based stage assigns fixed coupling orders for three trips and a semi-fixed coupling order for one trip. Based on the results of the platform-based stage, the network-based stage further assigns the coupling orders for another two trips and the semi-fixed trip has also been further fixed through coupling order propagation. There is one trip formed by a coupling operation of two units and one of them involves re-platforming operation. Its feasible coupling order and re-platforming time boundary are also processed at the network-based stage.

Dataset D2 runs three iterations to find a conflict-free solution, and detailed running information is shown in Table 5.

We can notice that the solution fleet size of each iteration is the same and the objective function has a slight increase as we expected for compromising of

producing a conflict-free schedule. In this process, four conflicts are detected during the platform-based stage,in which two of them are locally resolvable. Hence, only two station level constraints (crossing linkages) are added back to Phase I which are shown in the fifth column in Table 5. Since all the conflicts are detected during the platform-based stage, the arc and flow changing through two iterations are visualized in Fig. 14 where the red arrows represent crossing linkages. The conflicts are resolved by some arc and unit type changes for T14 and T20.

The experimenting information for dataset D3 is shown in Table 6. For the solution from Phase I of each iteration, there was no conflict detected at the platform-based stage but one conflict was detected at the network-based stage. In the first solution, a coupling-order collision is located in the arc set $\{20, 27, 38, 40, 47, 48\}$, which is converted as constraint (29) such that all the solutions that contain these arcs are eliminated.

With constraint (29), *RS-Opt* delivers a new solution, which still contains a coupling-order collision in $\{34, 43, 51, 52, 61, 62\}$, converted as constraint (30). *RS-Opt* has been launched again with constraints (29) and (30) to deliver another solution which is finalized as conflict-free with assigned coupling orders.

$$y_{20} + y_{27} + y_{38} + y_{40} + y_{47} + y_{48} \leq 5 \tag{29}$$

Table 5 Information on each iteration for D2

Iteration	Objective	Fleet	Time	Conflicts	New constraint arc set
1	8.09951	4X,4Y	10.1s	0	No
2	8.09954	4X,4Y	11.3s	2	{80,85}
3	8.10041	4X,4Y	9.4s	2	{83,85}

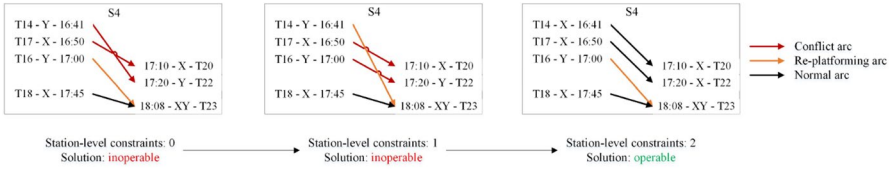

Fig. 14 Station level arc-flow changing

Table 6 Information on each iteration for D3

Iteration	Objective	Fleet	Time	Conflicts	New constraint arc set
1	6.08601	3X,3Y	11.5s	0	No
2	6.08603	3X,3Y	10.9s	1	{20,27,38,40,47,48}
3	7.08598	3X,4Y	207.1s	1	{34,43,51,52,61,62}

$$y_{34} + y_{43} + y_{51} + y_{52} + y_{61} + y_{62} \leq 5 \tag{30}$$

Note that the total number of train units used in the final solution has increased by one unit *Y*. It is because the problem size is small and there are not many other feasible solutions with the same fleet size. For the real-world dataset, increasing some units to find a feasible solution at the station level would not be frequent, since there would have been more feasible solutions corresponding to the minimum number of train units.

For dataset D4, two individual conflicts were detected for the solution without any station level constraints such that two new constraints are added to the next iteration. However, the new solution turns out two overlapping conflicts and the conflict arc sets are {20,27,38,39,47,48} and {34,39,51,52}. As it is seen here, arc 39 belongs to two conflicts. Hence, two strategies have been taken to tackle this type of overlapping conflict. One is to treat them as two constraints since no matter which sets of arcs are selected simultaneously will definitely invalid the solution. For this strategy, two more iterations are needed to find a feasible solution. The other is to consider them as an integrated constraint since it is possible to have a feasible solution if the flows on any detected incompatible arc set have a slight change. The experiment results show the solution following this strategy is conflict-free directly after adding this integrated constraint. This experiment shows that the two strategies can produce feasible results but the second strategy gives more flexibility in terms of arc selection, which means the second strategy is not as over-tight as the first strategy. Compared to the first-strategy solution, this solution has higher unit-usage efficiency in terms of fleet size, and the solution graph is much simpler. We also tried to combine these two strategies, but the results are still not as efficient as the second strategy.

Table 7 shows information of arc selection of all datasets, where Arcs(1) represent the number of arcs selected in the solution without station level constraints and Arcs(2) is on behalf of the number of arcs chosen in the final conflict-free solution.

The arc overlapping percentage between Arcs(1) and Arcs(2) is very high. Considering the solution structure, we notice that the conflict-free solution only changes a small portion of arcs around the detected arcs and tries to reserve most of the other low-cost arcs. The conflict-free optimal solution is always found within reasonable time and iterations. Besides, the feasible coupling orders and flexible timings can be determined simultaneously.

Table 7 Information on arc selection for D1, D2, D3 and D4

Dataset	Arcs(1)	Arcs(2)	Overlapping (%)	Total conflicts	Iterations
D1	24	24	100	0	1
D2	31	31	87.1	4	3
D3	27	28	85.2	2	3
D4-strategy 1	27	28	85.2	4	5
D4-strategy 2	27	31	92.6	8	3

7.3 Results of real-world datasets

The experiments on the artificial datasets endorse the correctness and effectiveness for the proposed method. The artificial datasets are specifically designed to capture all the features described in this research and there are only very limited numbers of feasible solutions corresponding to the optimal number of train units. These two reasons lead the artificial datasets to be more complicated to obtain a conflict-free solution at the station level. In other words, real-world datasets may be easier (less iterations) to be solved.

Table 8 gives a summary of the results of datasets *Sub*1 and *Sub*2 from Transpennine Express (see Sect. 7.1). These two datasets do not involve any coupling or decoupling operations, thus, there is no coupling-order conflict detected in the coupling order propagation process.

The conflict-free solution schedule of *Sub*1 needs two iterations to be finalized, and the details of each iteration are shown in Table 9. At the first iteration, three platform conflicts are detected, in which the first two conflicts at the platform 1 of MNCRIAP (Manchester Airport station) are locally resolvable by taking advantage of the interchangeability between the same type of train unit. However, the third platform conflict at the platform 1 of LVRPLSH (Liverpool Lime Street station) is unresolvable, marked as bold and underlined.

Thus, this conflict is converted as a new constraint (31) added to *RS-Opt* to seek another solution. In the new iteration, *RS-Opt* gives a solution with the same fleet size but a slightly higher objective function value. At Phase II, another four platform conflicts (located at NWCSTLE_2 and LVRPLSH_1) are detected. As all of them are locally resolvable, the locally resolved solution is the final conflict-free schedule.

Table 8 Information on each iteration for *Sub*1 and *Sub*2

Data	Iteration	Objective	Fleet size	Time (s)	Conflicts
*Sub*1	1	31.839	20, 21	16	3
	2	32.2474	20, 21	26696	4
*Sub*2	1	34.0061	11, 17	39	3

Table 9 Iterative conflicts of *Sub*1

Iteration	Location	Arc ID	Remark	Local resolvable
1	MNCRIAP_1	{467,525}	Crossing linkages	Yes
	MNCRIAP_1	{655,667}		Yes
	LVRPLSH_1	{314,357,430}		**No**
2	NWCSTLE_2	{123,351,154}	Crossing linkages	Yes
	NWCSTLE_2	{556,590}		Yes
	NWCSTLE_2	{220,293}		Yes
	LVRPLSH_1	{605,665}		Yes

$$y_{314} + y_{357} + y_{430} \leq 2 \tag{31}$$

However, the solution of *Sub*2 obtained at the first iteration could be finalized directly because only three local-resolvable platform conflicts located at MNCRIAP_1 are detected, seen in Table 10.

The solution of *Sub*3 from the network level was finalized as conflict-free at the station level, where three coupling operations were assigned with feasible coupling orders, as shown in Figs. 15 and 16. The coupling operation in Fig. 15 is operated at Edinburgh Waverley station. The coupling order in this figure is not important because of the interchangeability between the same type of train unit.

Figure 16 contains one coupling operation and one decoupling operation, operated at GLGC (Glasgow Central Station). The coupling order of trip 1M96FP at its origin is [$c350$, $c185$], formed by the coupling operation between the train units serving trips 1S35LP and 1S35LL. The coupling order of trip 1S71LP at its destination is [$c185$, $c350$], requested by the decoupling operation involving 1M94FA and 1M94LL. As 1M96FP and 1S71LP have no reversal en-route, the assigned coupling orders are the same during the whole journeys.

The coupling orders between 1M96FP and 1S71LP are compatible as this arc is operated at a dead-end platform of MNCRIAP (Manchester Airport station) such that there must be a reversal operation. Besides, the solution has five re-platforming operations that do not involve any coupling/decoupling operations. Therefore, they are fixed by assigning the same platforms to the involved trips.

Table 10 Platform conflicts of *Sub*1

Iteration	Location	Arc ID	Remark	Local resolvable
1	MNCRIAP_1	{501,558}	Crossing linkages	Yes
	MNCRIAP_1	{1511,1542}		Yes
	MNCRIAP_1	{1603,1628}		Yes

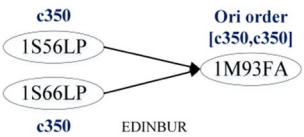

Fig. 15 Coupling operation at Edinburgh Waverley station

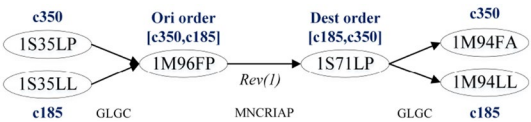

Fig. 16 Coupling order propagation

8 Conclusions and future research

The network level model has the limitation of ignoring station level constraints, which leads to an incomplete solution. This defect restricts the operability when it is implemented at the station level because of a set of undecided factors, such as the coupling order impacted by station layouts, timings, and unit movement directions. This research scrutinizes potential problems of a solution at the network level. A mathematical model with the capability of eliminating station level conflicts is proposed, in which two types of alternative station level constraints are introduced in detail. Based on the research about the network level of the TUSP, an adaptive approach is proposed to consider the station level constraints efficiently and to produce a complete and operable solution by systematically analyzing and iteratively adding station level conflict constraints. Through this method, the linkages given by the basic network level model can be verified and station level conflicts can be detected and resolved. In addition, the conflict-free coupling orders can also be determined. Both synthetic and real-world datasets are tested and the results are promising.

In our future research, more features and connections between the network level and station level will be investigated. The setup and configuration for testing and refining our models, especially on implementing the conflict constraints in *RS-Opt*, are substantial ongoing tasks, which need to be carefully analyzed with artificial and real-world datasets. More details and results will be reported in a future paper.

Acknowledgements This research is supported by an Engineering and Physical Sciences Research Council (EPSRC) project EP/M007243/1. We would like to also thank First TransPennine Express, Abellio (UK), and Tracsis PLC for their kind and helpful collaboration.

References

Balas E, Jeroslow R (1972) Canonical cuts on the unit hypercube. SIAM J Appl Math 23(1):61–69

Cacchiani V, Caprara A, Toth P (2010) Solving a real-world train-unit assignment problem. Math Program 124(1–2):207–231

Cacchiani V, Caprara A, Maróti G, Toth P (2013a) On integer polytopes with few nonzero vertices. Oper Res Lett 41(1):74–77

Cacchiani V, Caprara A, Toth P (2013b) A Lagrangian heuristic for a train-unit assignment problem. Discrete Appl Math 161(12):1707–1718

Copado-Mendez P, Lin Z, Kwan R (2017) Size limited iterative method (SLIM) for train unit scheduling. In: Proceedings of the 12th Metaheuristics International Conference, Barcelona, Spain

Freling R, Lentink RM, Kroon LG, Huisman D (2005) Shunting of passenger train units in a railway station. Transport Sci 39(2):261–272
Haahr J, Lusby RM (2017) Integrating rolling stock scheduling with train unit shunting. Eur J Oper Res 259(2):452–468
Huisman D, Kroon LG, Lentink RM, Vromans MJCM (2005) Operations research in passenger railway transportation. Stat Neerl 59(4):467–497
Kroon LG, Lentink RM, Schrijver A (2008) Shunting of passenger train units: an integrated approach. Transp Sci 42(4):436–449
Kwan RSK, Lin Z, Copado-Mendez PJ, Lei L (2017) Multi-commodity flow and station logistics resolution for train unit scheduling. In: Proceedings of the 8th multidisciplinary international conference on scheduling: theory and applications, MISTA, pp 321–324
Lei L, Kwan RSK, Lin Z, Copado-Mendez PJ (2017) Station level refinement of train unit network flow schedules. In: 8th International Conference on Computational Logistics, Southampton, UK
Lin Z, Kwan RSK (2013) An integer fixed-charge multicommodity flow (FCMF) model for train unit scheduling. Electron Notes Discrete Math 41:165–172
Lin Z, Kwan RSK (2014) A two-phase approach for real-world train unit scheduling. Public Transp 6(1–2):35–65
Lin Z, Kwan RSK (2016a) A branch-and-price approach for solving the train unit scheduling problem. Transp Res Part B Methodol 94:97–120
Lin Z, Kwan RSK (2016b) Local convex hulls for a special class of integer multicommodity flow problems. Comput Optim Appl 64(3):881–919
Lin Z, Barrena E, Kwan RSK (2017) Train unit scheduling guided by historic capacity provisions and passenger count surveys. Public Transp 9(1–2):137–154
Lin Z, Kwan RSK (2017) Multicommodity flow problems with commodity compatibility relations. In: ITM Web of Conferences, Volume 14, 2017 The 12th international conference applied mathematical programming and modelling—APMOD 2016, EDP Sciences, vol 14
Tomii N, Zhou LJ, Fukumura N (1999) An algorithm for station shunting scheduling problems combining probabilistic local search and PERT. In: International conference on industrial, engineering and other applications of applied intelligent systems. Springer, pp 788–797
Wikipedia (2019) https://en.wikipedia.org/wiki/Passenger_rail_franchising_in_Great_Britain